Lecture 21
Verification Condition Generation (I)
Instructor: Hossein Hojjat

October 23, 2017
Hoare Rules: Summary

\[\vdash \{A[x \mapsto e]\} \ x := e \ {A}\]
\[\vdash \{A \land b\} \ c_1 \ {B} \quad \vdash \{A \land \neg b\} \ c_2 \ {B}\]
\[\vdash \{A\} \ \text{if} \ b \ \text{then} \ c_1 \ \text{else} \ c_2 \ {B}\]
\[\vdash \{A \land b\} \ c \ {A}\]
\[\vdash \{A\} \ \text{while} \ b \ \text{do} \ c \ {A \land \neg b}\]
\[\vdash \{A\} \ c_1 \ {C} \quad \vdash \{C\} \ c_2 \ {B}\]
\[\vdash \{A\} \ c_1 \ ; \ c_2 \ {B}\]
\[\vdash A' \Rightarrow A \quad \vdash \{A\} \ c \ {B} \quad \vdash B \Rightarrow B'\]
\[\vdash \{A'\} \ c \ {B'}\]
• Manually proving correctness is tedious
• We’d like to automate the tedious parts of program verification
• Idea: Assume an oracle gives loop invariants - we can then automate the rest of the reasoning
• This oracle can either be a human or a static analysis tool
  • (e.g., abstract interpretation)
Automated Verification

- Example specs: safety (no crashes), absence of arithmetic overflow, complex behavioral property (e.g., “sorts an array”)
• Automating Hoare logic is based on generating verification conditions (VC)
• A verification condition is a formula $\phi$ such that program is correct iff $\phi$ is valid
• Deductive verification has two components:
  1. Generate VC’s from source code
  2. Use theorem prover to check validity of formulas from step 1
Two ways to generate verification conditions: forwards or backwards.

- A forwards analysis starts from precondition and generates formulas to prove postcondition.
- Forwards technique computes strongest postconditions (sp).
- In contrast, backwards analysis starts from postcondition and tries to prove precondition.
- Backwards technique computes weakest preconditions (wp).
Strongest Postconditions

- Some valid Hoare Triples

\[
\begin{align*}
\{ x = 5 \} & \quad x := x + 5 & \{ \text{true} \} \\
\{ x = 5 \} & \quad x := x + 5 & \{ x > 0 \} \\
\{ x = 5 \} & \quad x := x + 5 & \{ x = 10 \lor x = 5 \} \\
\{ x = 5 \} & \quad x := x + 5 & \{ x = 10 \}
\end{align*}
\]

- All are valid but \( x = 10 \) is the most useful one

- Strongest postcondition

- If \( \{ P \} \ S \{ Q \} \) and for all \( Q' \) such that \( \{ P \} \ S \{ Q' \} \), \( Q \Rightarrow Q' \), then \( Q \) is the strongest postcondition of \( S \) with respect to \( P \)

- check: \( x = 10 \Rightarrow \text{true} \)
- check: \( x = 10 \Rightarrow x > 0 \)
- check: \( x = 10 \Rightarrow x = 10 \lor x = 5 \)
- check: \( x = 10 \Rightarrow x = 10 \)
Weakest Preconditions

• Some valid Hoare Triples (assume an extension of IMP with division)

\{ x = 5 \land y = 10 \} \quad z := x/y \quad \{ z < 1 \}
\{ x < y \land y > 0 \} \quad z := x/y \quad \{ z < 1 \}
\{ y \neq 0 \land x/y < 1 \} \quad z := x/y \quad \{ z < 1 \}

• All are valid but \( y \neq 0 \land x/y < 1 \) is the most useful one
• It allows us to invoke the program in the most general condition
  • Weakest precondition
• If \( \{ P \} \ S \ \{ Q \} \) and for all \( P' \) such that \( \{ P' \} \ S \ \{ Q \} \), \( P' \Rightarrow P \),
  then \( P \) is the weakest precondition \( \text{wp}(S, Q) \) of \( S \) with respect to \( Q \)
Weakest Preconditions

- Arc 1 produces a final state not satisfying $Q$  
- Arc 2 produces a final state satisfying $Q$  
- Arc 3 gets into a loop and produces no final state
Backwards Method

- Idea: Suppose we want to verify Hoare triple $\{P\} \ S \ \{Q\}$
- Start with $Q$ and go backwards, compute formula $wp(S, Q)$
- We can dually define forward method using $sp$
  - (won’t discuss in lecture)
- Weakest preconditions are defined inductively following Hoare’s rules

- $wp(x := E, Q) = Q[x \mapsto E]$
- $wp(c_1; c_2, Q) = wp(c_1, wp(c_2, Q))$
- $wp(\text{if } b \text{ then } c_1 \text{ else } c_2, Q) = b \rightarrow wp(c_1, Q) \land \neg b \rightarrow wp(c_2, Q)$

- Rule for conditional says:
  “If $b$ holds, $wp$ of then branch must hold; otherwise, $wp$ of else branch must hold”
Exercise

Consider the following code $S$

$$x := y + 1; \text{ if } x > 0 \text{ then } z := 1 \text{ else } z := -1$$

• What is $wp(S, z > 0)$?
• What is $wp(S, z \leq 0)$?

$\begin{align*}
&\text{• } wp(x := E, Q) = Q[x \mapsto E] \\
&\text{• } wp(c_1; c_2, Q) = wp(c_1, wp(c_2, Q)) \\
&\text{• } wp(\text{if } b \text{ then } c_1 \text{ else } c_2, Q) = b \rightarrow wp(c_1, Q) \land \neg b \rightarrow wp(c_2, Q)
\end{align*}$
Exercise

• Consider the following code $S$

\[
x := y + 1; \text{ if } x > 0 \text{ then } z := 1 \text{ else } z := -1
\]

• What is $\text{wp}(S, z > 0)$? $y \geq 0$

• What is $\text{wp}(S, z \leq 0)$?

• $\text{wp}(x := E, Q) = Q[x \mapsto E]$

• $\text{wp}(c_1; c_2, Q) = \text{wp}(c_1, \text{wp}(c_2, Q))$

• $\text{wp}(\text{if } b \text{ then } c_1 \text{ else } c_2, Q) = b \rightarrow \text{wp}(c_1, Q) \land \neg b \rightarrow \text{wp}(c_2, Q)$
Exercise

- Consider the following code $S$

  \[
  x := y + 1; \text{ if } x > 0 \text{ then } z := 1 \text{ else } z := -1
  \]

- What is $wp(S, z > 0)$? $y \geq 0$
- What is $wp(S, z \leq 0)$? $y < 0$

- $wp(x := E, Q) = Q[x \mapsto E]$
- $wp(c_1; c_2, Q) = wp(c_1, wp(c_2, Q))$
- $wp(\text{if } b \text{ then } c_1 \text{ else } c_2, Q) = b \rightarrow wp(c_1, Q) \land \neg b \rightarrow wp(c_2, Q)$
It is convenient to extend the language with the following statements:

- **havoc** $x$ change $x$ to an arbitrary value
- **assert** $b$ if $b$ holds, terminate; otherwise, exit with error
- **assume** $b$ if $b$ holds, terminate; otherwise, block

```
{ x=x0 and y=y0 }
  z = x;
  x = y;
  y = z;
{ y=x0 and x=y0 }
```

```plaintext
assume x = x0;
assume y = y0;
z = x;
x = y;
y = z;
assert x = x0;
assert y = y0;
```
Assert, Assume and Havoc

- $\text{wp}(\text{assert } b, Q) = b \land Q$
- For $Q$ to be true after, $b$ must also be true before, because otherwise we won’t get past the assert
- $\text{wp}(\text{assume } b, Q) = b \rightarrow Q$
- If $b$ is not true, we don’t care if $Q$ is satisfied
- $\text{wp}(\text{havoc } x, Q) = \forall y. Q[x \mapsto y]$
Weakest Preconditions for Loops

• Let’s start from the equivalence
  \[ \text{while } b \text{ do } c = \text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip} \]

• Assume \( W = \text{wp}(\text{while } b \text{ do } c, Q) \)

• It must be that: \( W = (b \Rightarrow \text{wp}(c, W) \land \neg b \Rightarrow Q) \)

• But this is a recursive equation!
Weakest Preconditions for Loops

- Can be computed by using an invariant

\[ \text{wp}(\text{while } b \text{ do } c, Q) = \]
\[ I \land \forall \vec{y}. \left( (I \land b \Rightarrow \text{wp}(c, I)) \land (I \land \neg b \Rightarrow Q) \right)[\vec{x} \mapsto \vec{y}] \]

- where \( \vec{x} = x_1, \ldots, x_n \) are variables modified in \( c \) and \( I \) is the loop invariant
We encode the meaning of `while` statement as following:

```plaintext
while b do c ≡
assert I;
havoc x₁, ..., xₙ;
assume I;
if (b) then
{
c
assert I;
assume false;
}
```

- `assert I`: Loop invariant is checked on entry
- We “fast forwards” to an arbitrary iteration by setting the variables to arbitrary values
- In that arbitrary iteration we check that the loop condition is defined
- Either we perform one more iteration or terminate the loop
- `assume false` indicates that the remainder of program is not analyzed immediately after an arbitrary loop iteration
- Analysis only proceeds under the successful termination